OpenJTAG USER MANUAL

Version 1 Revision 4

Dynalith Systems Co., Ltd.

www.dynalith.com
contact@dynalith.com

July. 2008

Copyright

Copyright © 2007-2008 Dynalith System Co., Ltd.
All rights reserved. No part of this publication may be reproduced or used in any form or by any means including graphic, electronic, or mechanical, or by photocopying, recording, taping, or information storage and retrieval systems, without written permission of Dynalith System Co., Ltd.

Trademarks

Dynalith is registered trademark of Dynalith System Co., Ltd.
The Dynalith logo, iPROVE, the iPROVE logo, iProveBaseBoard, iProveAppBoard, BILA, DPP, iGnite, RTL BILA, RTL SCAN iTUTOR, iNSPIRE, iVORY, OpenIDEA, iNCITE, iNCITE-AVREM, and OpenJTAG are trademarks of Dynalith System Co., Ltd.
All other brand or product names may be trademarks or registered trademarks of their respective holders.

Dynalith System Co., Ltd.
14-2, Yangjae-dong, Taejin Bldg. 2F,
Seocho-ku, Seoul 137-888, Korea

Dynalith Systems R&D Center
335 Gwahangno (373-1 Kusong-Dong), Yusong-Gu, CHiPS B/D, KAIST
Daejon, 305-701, Korea

Tel: +82-2-556-0020
Fax: +82-2-556-2252
E-mail: contact@Dynalith.com
URL: http://www.Dynalith.com

Table of Contents

About this manual ix
CHAPTER1: Introduction 1-1
What is OpenJTAG 1-1
OpenJTAG structure.....1-1
OpenJTAG environment.....1-2
Related documents 1-2
CHAPTER2: Installation for PC Windows XP 2-1
Prerequisites and what to install 2-1
Installation steps 2-1
Software installation.....2-1
Standard installation directory structure 2-2
Environment variable 2-3
OpenJTAG software uninstallation 2-3
Device driver installation.....2-4
Hardware connection.....2-7
20-pin male connector for JTAG 2-7
20-to-8-wire adapter board 2-8
9-pin male connector 2-9
CHAPTER3: Getting started with OpenJTAG API 3-1
Reading identification register.....3-1
Programming with OpenJTAG API 3-2
How to compile with Visual Studio 6.0 3-3
How to compile with GNU GCC 3-10
CHAPTER4: OpenJTAG API and Programming 4-1
OpenJTAG control API 4-1
int oj_init(int config); 1
int oj_close(void); 1
int oj_set_speed(float clk); 1
Bit-wise control API.....4-2
int oj_write(unsigned int tall); 2
int oj_read(unsigned int* tall); 2
Bit-sequence control API.....4-2
int oj_cycle_out (unsigned int num, unsigned int trstn, unsigned int tms, unsigned int tdi); 2
int oj_cycle (unsigned num, unsigned trstn, unsigned tms, unsigned tdi, unsigned *tdo); 2
int oj_cycle_wide (unsigned int num, unsigned int *trstn, unsigned int *tms, unsigned int *tdi, unsigned int *tdo); 3
Bit-sequence control example 4-3
Programming with OpenJTAG API.....4-4
OpenJTAG header file 4-4
OpenJTAG library file 4-4

Visual Studio 6.0 for Windows XP 4
GNU gcc for Linux 4
CHAPTER5: GDB Server 5-1
Introduction to overall environment 5-1
Prerequisite 5-2
Steps to setup.....5-3
OpenJTAG setup 5-3
Run GDB server 5-5
Run GDB debugger 5-7
CHAPTER6: OpenJTAG GUI 6-1
OpenJTAG GUI.....6-1
Diagnostic tab.....6-2
Detect OpenJTAG 6-2
Diagnostics 6-2
Scan Device 6-2
Server tab.....6-3
Host IP field 6-3
Socket Port field 6-3
Server field 6-4
Start and stop button 6-4
Debugger tab 6-4
Debugger field 6-5
Option field 6-6
Command field 6-6
Execute file field 6-6
Start and stop button 6-6
Debugger GUI tab.....6-6
GUI program field 6-6
Option field 6-7
Command field 6-7
Start and stop button 6-7
How to use OpenJTAG for BA12.....6-8
How to use OpenJTAG for OpenRISC 6-9
Index 1

List of Figures

OpenJTAG 1-1
OpenJTAG structure 1-1
OpenJTAG software installation steps 2-2
Uninstall entry 2-4
OpenJTAG 20-pin male connector for JTAG 2-7
Conceptual schematic of OpenJTAG JTAG signals 2-8
OpenJTAG 8-wire cable 2-8
20-to-8-wire adapter board with 8-wire cable 2-9
9-pin male connector 2-9
One-cycle operation 4-3
Multi-cycle operation 4-4
Debugging environment around OpenJTAG 5-1
OpenJTAG GUI Layout 6-1
OpenJTAG Diagnostics tab 6-2
OpenJTAG Server tab 6-3
Ready-to-run debugger server list 6-4
OpenJTAG Debugger tab 6-5
Ready-to-run debugger 6-5
OpenJTAG Debugger GUI tab 6-6
Ready-to-run debugger GUI 6-7
Debugging environment with BA12 6-8
Debugging environment with OpenRISC 6-9

List of Tables

OpenJTAG environment 1-2
OpenJTAG directory structure 2-3

About this manual

This manual is about OpenJTAG.

Typographical conventions

The following typographical conventions are used in this manual:

- `Courier` font indicates messages, prompts, and program files that the system displays. It also indicates literal commands that you enter in a syntactical statement. However, braces "{ }" in Courier are not literal and square brackets "[ ]" in Courier are literal only in the case of bus specifications, such as bus[7:0]. Courier also indicates commands that you select from a menu.

  `File` → `Open`
- Italic font denotes the following items
  - Variable in a syntax statement for which you must supply values
  - Emphasis within text
- Square brackets "[ ]" indicate optional entry or parameter. However, in bus specifications, such as bus[7:0], they are mandatory.
- Braces "{ }" enclose a list of items from which you must choose one or more.

  type=reset | clock | VDD | VSS
- A vertical bar | separates items in a list of the choice.

  type=reset | clock | VDD | VSS

Signal name conventions

- Use square bracket [ ] for bus signals, such as ADDR[7:0].
- Use postfix 'b' or 'n' for active-low signals, such as RESETb.

Technical Support

For technical support, go to www.dynalith.com or send e-mail to contact@dynalith.com.

Revision history

The following shows the revision history of this manual.

- July. 18, 2008: Version 1 Revision 4 by K.S. Jeon.
- June. 19, 2008: Version 1 Revision 3 by K.S. Jeon.
- Feb. 18, 2008: Version 1 Revision 2 by Ando Ki and K.S. Jeon.
- Jan. 5, 2008: Version 1 Revision 1 by Ando Ki.
- Oct. 18, 2007: Start of documentation by Ando Ki.

Chapter 1

Introduction

1.1 What is OpenJTAG

OpenJTAG provides an easy to use channel between application program and JTAG port, i.e., Boundary Scan TAP (Test Access Port), where USB 2.0 is used as a physical channel.

Note: In this document, 'JTAG' (Joint Test Action Group) and 'Boundary Scan' are used interchangeably.

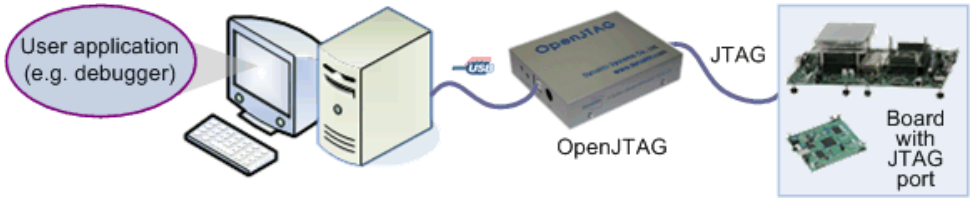


Figure 1-1: **OpenJTAG**

1.2 OpenJTAG structure

OpenJTAG consists of hardware and software; the hardware connects a USB to the JTAG TAP and the software consists of device driver and API.

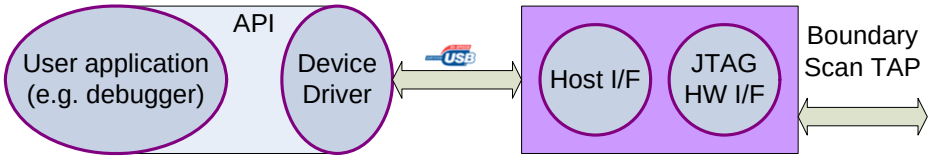


Figure 1-2: **OpenJTAG structure**

1.3 OpenJTAG environment

Table 1-1: **OpenJTAG environment**

Platform	Windows XP	Linux
Programming environment	Visual Studio 6.0 GNU GCC	GNU GCC
Supporting programs	GNU GDB	GNU GDB

1.4 Related documents

IEEE Std. 1149.1-2001, IEEE Standard Test Access Port and Boundary-Scan Architecture, Test Technology Standards Committee of the IEEE Computer Society.

dynalith

Chapter 2

Installation for PC Windows XP

This chapter describes installation of the environment for the Open-JTAG.

2.1 Prerequisites and what to install

OpenJTAG package consists of hardware and software.

- The hardware is comprised of OpenJTAG, 20-to-8-wire adapter board, 8-wire cable and USB cable.
- The software contains USB device driver and OpenJTAG API library. In addition to this, it may contains various debugger server programs depending on software package.

Thus, the following should be prepared before install OpenJTAG.

- Computer platform: PC Windows XP
- Platform requirement: Pentium IV or later with main memory of more than 512MByte and free hard-disk space of more than 100Mbyte.
- Platform with USB 2.0

2.2 Installation steps

Software installation should be done prior to hardware connection since device driver is required during hardware connection.

- step 1: software installation
- step 2: device driver installation
- step 3: hardware connection

2.3 Software installation

OpenJTAG software will be given one of several forms.

- Download from Dynalith Systems web-page, i.e., http://www.dynalith.com.,
- USB memory stick, or
- Compact Disk.

OpenJTAG hardware should be disconnected from the host computer during the initial software installation.

Installation will start by invoking installation software, which will be something like 'OpenJTAG_V1_1_B01.exe'.

Installing steps are so conventional that just follow the on screen instructions as shown in Figure 2-1. The default installation directory will be C:\OpenJTAG, but you can change if you want while installation.

Default installation directory is as follows.

- For Windows XP: C:\OpenJTAG

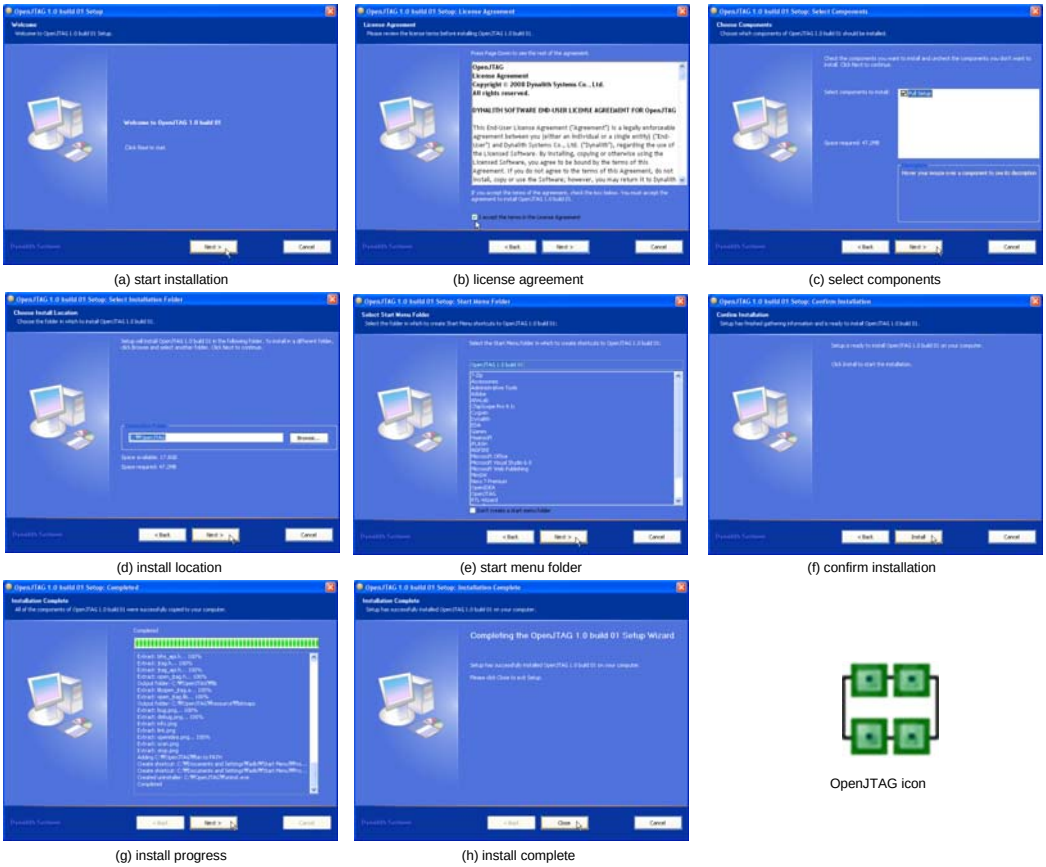


Figure 2-1: **OpenJTAG software installation steps**

2.3.1 Standard installation directory structure

After installation, OpenJTAG software builds its directory in the installation directory as given in Table 2-1, where installation directory will be C:\OpenJTAG by default.

Table 2-1: **OpenJTAG directory structure**

Directory		Remarks
bin		programs
doc		manuals
drv		device driver
example		example programs
hardware	eif_200	for OpenJTAG-200
	eif_400	for OpenJTAG-400
include		headers
lib	gcc-x.y.z	library for GCC version x.y.x
	mingw-a.b.c	library for GCC version a.b.c on MinGW
	visualc-6.0	library for Visual Studio 6.0
resource		GUI related resources

On successful installation, there will be an entry for OpenJTAG at Start->Programs menu as shown below.

When you allowed registration of short-cut icon during installation the OpenJTAG icon will appear on the desktop as follows.

2.3.2 Environment variable

OpenJTAG installer registers ‘OPENJTAG’ environment variable automatically and it directs where OpenJTAG is installed. Thus, $OPENJTAG will be C:\OpenJTAG by default.

2.3.3 OpenJTAG software uninstallation

To remove OpenJTAG software products from your system, use the Dynalith Systems un-installation program. Follow this procedure:

1. Quit OpenJTAG before trying to un-install it.

2. Start the un-installer.

Click the Start button of Windows, select ‘Programs’ menu, and move the cursor over the OpenJTAG entry. On the application menu that appears, click the entry for the un-installer as shown in Figure 2-2

Figure 2-2: **Uninstall entry**

2.4 Device driver installation

After installing OpenJTAG software, the first time the OpenJTAG is attached, Windows invokes the Found New Hardware wizard.

As the OpenJTAG and the host computer are connected each other through USB, the host computer requires USB device driver for the OpenJTAG hardware.

The device driver for the OpenJTAG resides in ‘drv’ directory under the OpenJTAG installation directory, e.g., C:\OpenJTAG\drv and it contains the following.

- ivory.inf
- ivory.sys

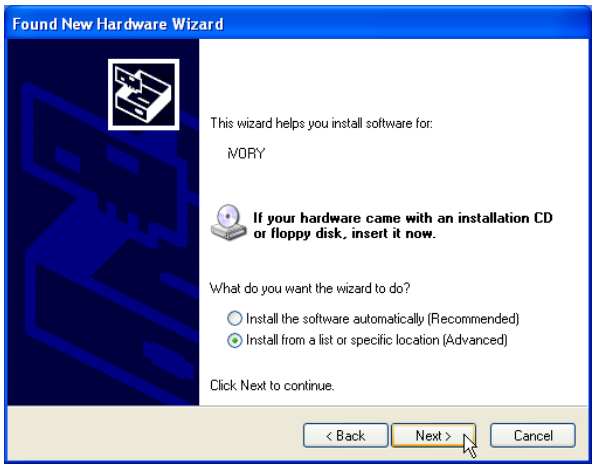
To install the OpenJTAG USB device driver:

1. Connect the OpenJTAG hardware to the host computer through the USB cable.
2. The hardware wizard will be invoked.

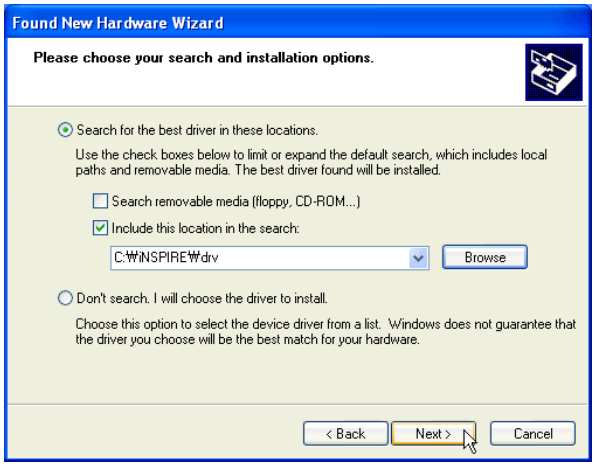
3. Select the ‘Yes, this time only’ and click ‘Next’.

Found New Hardware Wizard
Welcome to the Found New Hardware Wizard
Windows will search for current and updated software by looking on your computer, on the hardware installation CD, or on the Windows Update Web site (with your permission).
Read our privacy policy
Can Windows connect to Windows Update to search for software?
Yes, this time only
Yes, now and every time I connect a device
No, not this time
Click Next to continue.
< Back
Next >
Cancel

4. Select ‘Install from a list or specific location (Advanced)’ and click ‘Next>’


Found New Hardware Wizard
This wizard helps you install software for:
iVORY
If your hardware came with an installation CD or floppy disk, insert it now.
What do you want the wizard to do?
Install the software automatically (Recommended)
Install from a list or specific location (Advanced)
Click Next to continue.
< Back
Next >
Cancel

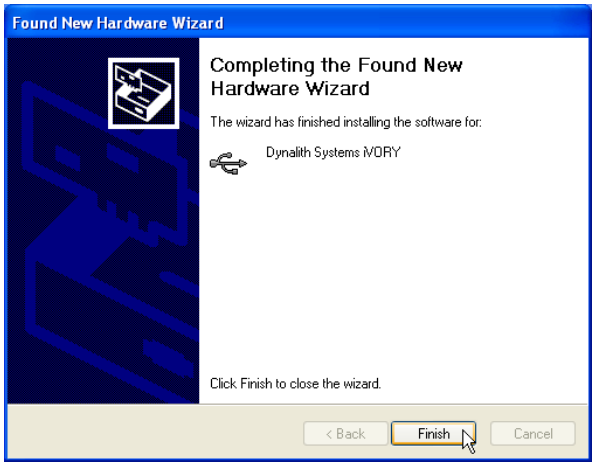
5. Specify the directory name where OpenJTAG device driver locates, e.g., C:\OpenJTAG\drv and click ‘Next>’


Found New Hardware Wizard
Please choose your search and installation options.
Search for the best driver in these locations.
Use the check boxes below to limit or expand the default search, which includes local paths and removable media. The best driver found will be installed.
Search removable media (floppy, CD-ROM...)
Include this location in the search:
C:₩iNSPIRE₩drv
Browse
Don't search. I will choose the driver to install.
Choose this option to select the device driver from a list. Windows does not guarantee that the driver you choose will be the best match for your hardware.
< Back
Next >
Cancel

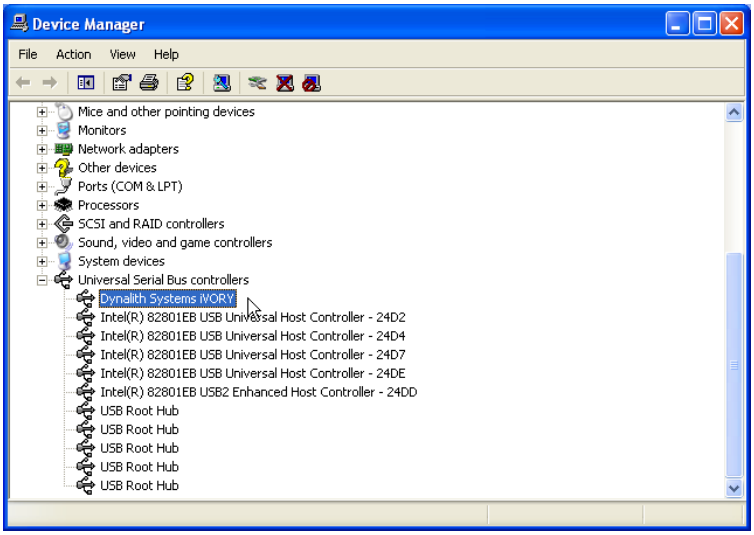
6. The hardware wizard will search for the device driver. (Note that you might encounter a warning message regarding ‘Win-

dows Logo testing' and just ignore it by clicking 'Continue Anyway'.)

7. Then the hardware wizard installs the OpenJTAG device driver.
8. Finally, the hardware wizard should report the completion of installation.



9. After installation, the device driver can be found from system hardware device manager, i.e., Start->Control Panel->System->Hardware->Device Manager, where a new entry of `Dynalith Systems iVORY' as one of 'Universal Serial Bus controllers' will be found.



Windows invokes the Found New Hardware wizard each time the OpenJTAG is plugged into a different physical USB port for the first time. Thus, you should follow the above device driver installation procedure.

2.5 Hardware connection

OpenJTAG provides two different connectors.

- 20-pin male connector for JTAG
  - ❑ 20-to-8-wire adapter
- 9-pin male connector for I2C and SPI (this is not supported yet)

2.5.1 20-pin male connector for JTAG

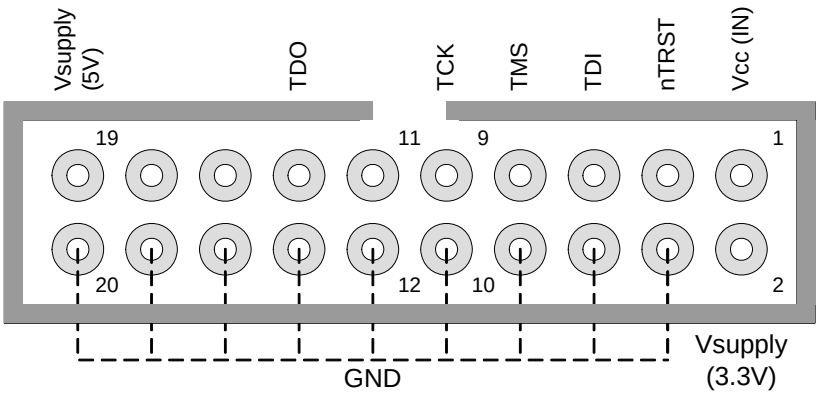
There are five JTAG pins, which are as follows

- nTRST (i.e., TRSTn): active low JTAG reset output in terms of OpenJTAG; ‘nTRST’ is not used for some cases.
- TCK: JTACK clock output in terms of OpenJTAG
- TMS: JTAG mode set output in terms of OpenJTAG
- TDI: JTAG data output in terms of OpenJTAG
- TDO: JTAG data input in terms of OpenJTAG

In addition to this three power pins and nine signal ground pins.

- Vcc (IN): High-level voltage input for JTAG signal
- Vsupply (3.3V): High-level voltage output for external board
- Vsupply (5V): High-level voltage output for external board
- GND: Signal ground pins

It should be noted that ‘VCC(IN)’ and ‘GND’ should be connected to the target JTAG port.



20-pin JTAG connector

Figure 2-3: **OpenJTAG 20-pin male connector for JTAG**

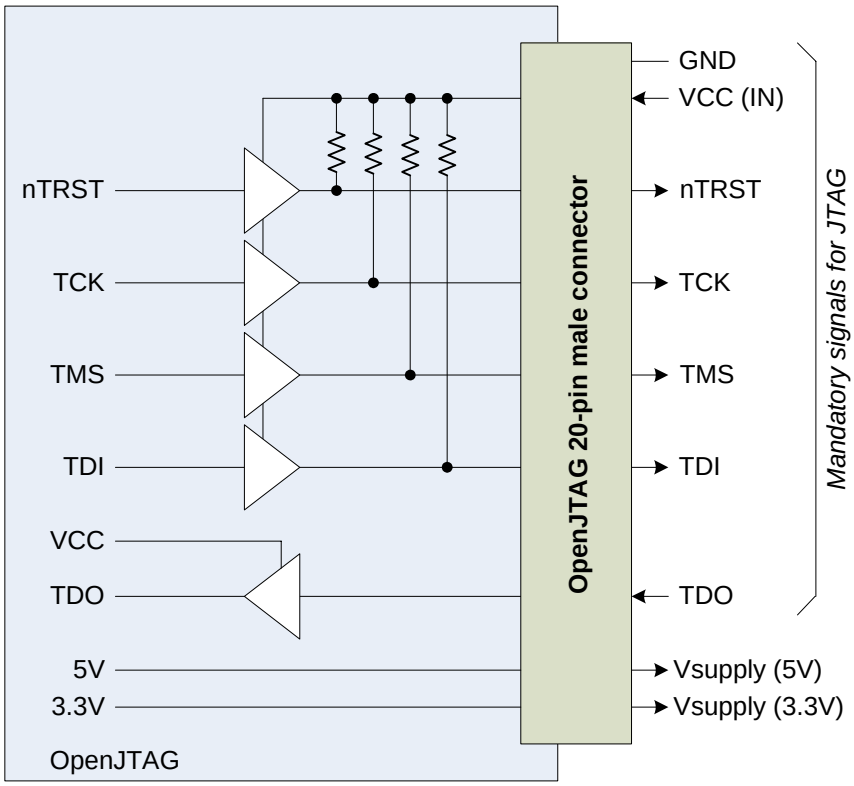


Figure 2-4: **Conceptual schematic of OpenJTAG JTAG signals**

2.5.2 20-to-8-wire adapter board

OpenJTAG also provides a 8-wire cable for a normal 6-pin JTAG connection.

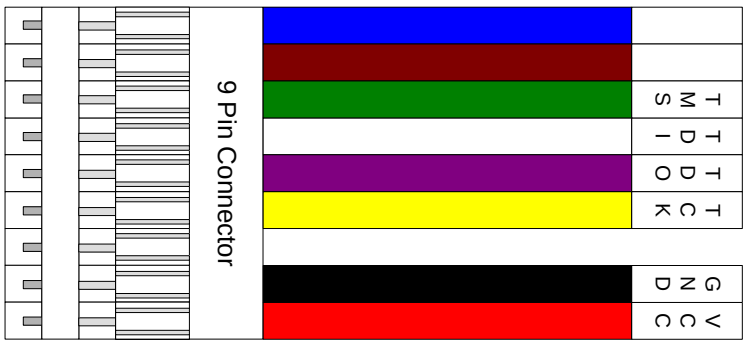


Figure 2-5: **OpenJTAG 8-wire cable**

In order to use 8-wire cable with 20-pin male connector, 20-to-8-wire adapter board is used.

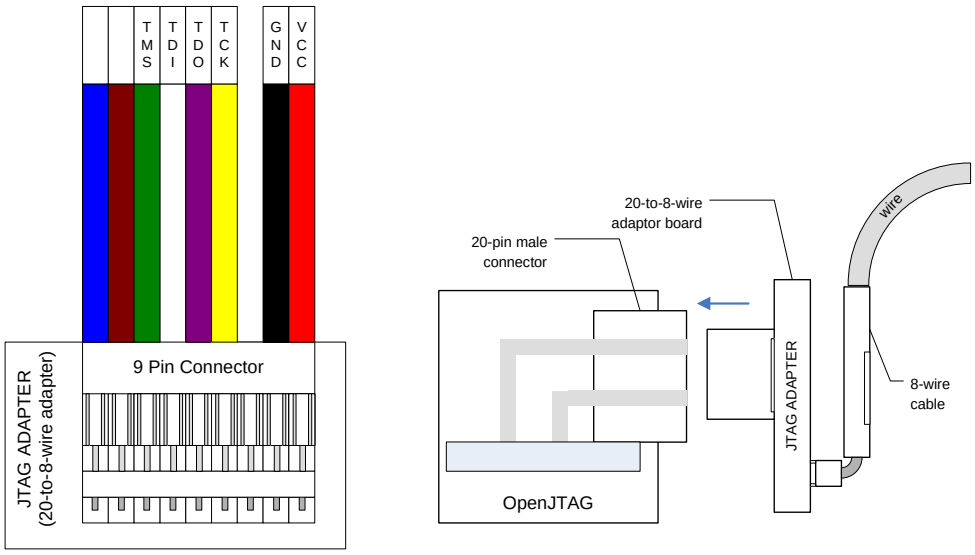


Figure 2-6: **20-to-8-wire adapter board with 8-wire cable**

2.5.3 9-pin male connector

OpenJTAG provides 9-pin male connector for future usage, which includes I2C and SPI.

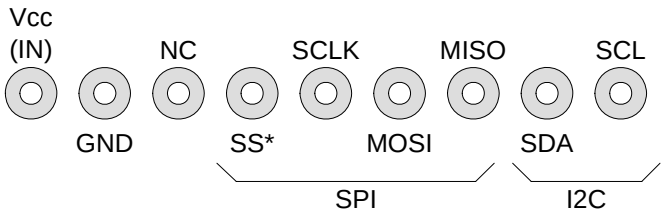


Figure 2-7: **9-pin male connector**

Chapter 3

Getting started with OpenJTAG API

This chapter addresses a quick tutorial of how to use OpenJTAG API to develop an application.

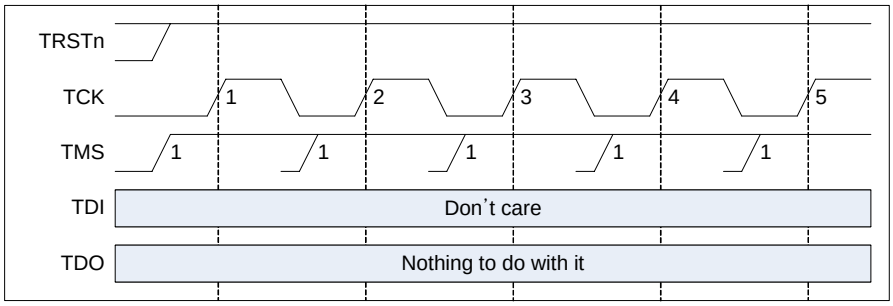
3.1 Reading identification register

JTAG test logic architecture contains a TAP (Test Access Port), a TAP controller, an instruction register and a group of test data registers.

When TAP controller is in ‘Test-Logic-Reset’ state, the instruction register will be set to give the IDCODE or BYPASS depending on the implementation of the TAP controller. Thus, the IDCODE can be read by shifting the data register while staying at ‘Shift-DR’ state.

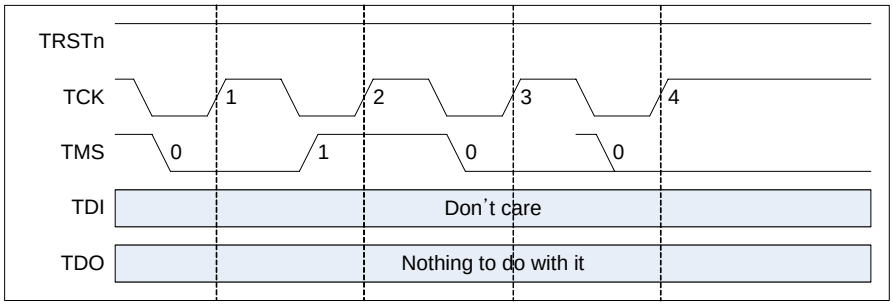
In other words, the following sequence gets IDCODE from JTAG TAP controller.

1. Go to ‘Test-Logic-Reset’ state
   - ❑ Holding TMS at high during at lease five rising edges of TCK.

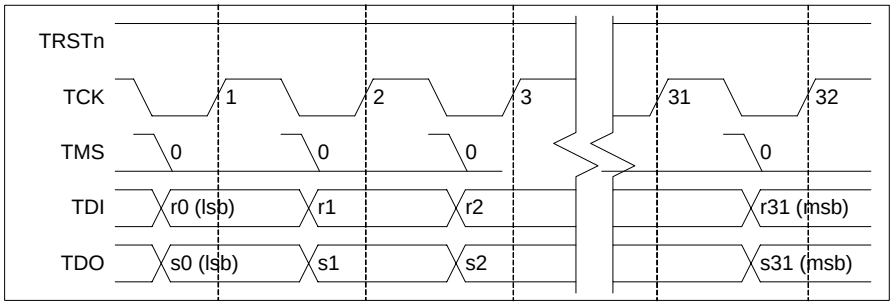


2. Go to ‘Shift-DR’ state through Run-Test-Idle, Select-DR-Scan, Capture-DR, respectively.

- ❑ Feeding ‘0010’ bit sequence (LSB first) in to TMS, in which TRSTn should be high and TDI is don’t care.



3. Providing 32 rising edges of TCK while TMS stays low.
   - ❑ Feeding ‘0x00FF’ bit sequence in to TDI, which is invalid IDCODE in order to detect end of sequence.



3.1.1 Programming with OpenJTAG API

OpenJTAG software library provides an API that controls TRSTn, TMS, TDI and TDO during a certain number of rising edge of TCK.

```
oj_cycle( unsigned int  num
        , unsigned int  trstn
        , unsigned int  tms
        , unsigned int  tdi
        , unsigned int *tdo);
```

This oj_cycle generates JTAG cycles specified by ‘num’, which can be up to 32. Each JTAG cycle uses the value of ‘trstn’, ‘tms’ and ‘tdi’ from its lsb (least significant bit). After completion, ‘tdo’ contains TDO values.

Thus, the IDCODE can be read by calling the following calls.

```
oj_cycle(5, 0xFFFFFFFF, 0x1F, 0x0, &tdo);
oj_cycle(4, 0xFFFFFFFF, 0x02, 0x0, &tdo);
oj_cycle(32, 0xFFFFFFFF, 0x00, 0xFF, &tdo);
```

In addition to these, there are two API which should be called in order to use OpenJTAG.

```
oj_init ( int config );
oj_close ( );
```

The oj_init() takes care of preparing necessary things for Open-JTAG and the oj_close() cleans up before closing OpenJTAG.

Thus, the program can be as follows.

```
#include <stdio.h>
#include "open_jtag.h"

int main(int argc, char* argv[]) {
     unsigned int tdo;
     if (oj_init(0)) return(1);
    oj_cycle(5, 0xFFFFFFFF, 0x1F, 0x0, &tdo);
    oj_cycle(4, 0xFFFFFFFF, 0x02, 0x0, &tdo);
    oj_cycle(32, 0xFFFFFFFF, 0x00, 0xFF, &tdo);
    printf("IDCODE: 0x%x\n", tdo);
    oj_close();
    return(0);
}
```

Where ‘open_jtag.h’ is a header file locates at ‘$(OpenJTAG)/include’ directory, where $(OpenJTAG) stands for the installation directory, say C:\OpenJTAG for Windows case.

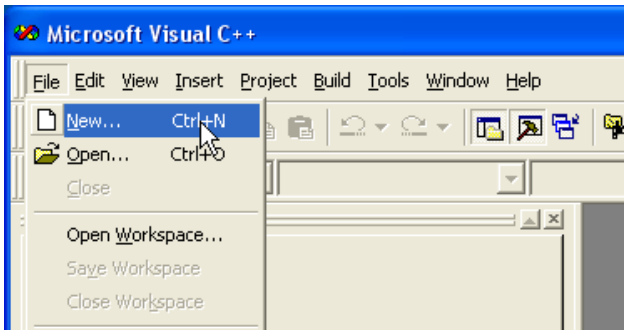
Note that this example is available at ‘$(OpenJTAG)/example/jtag_scan’ directory.

3.1.2 How to compile with Visual Studio 6.0

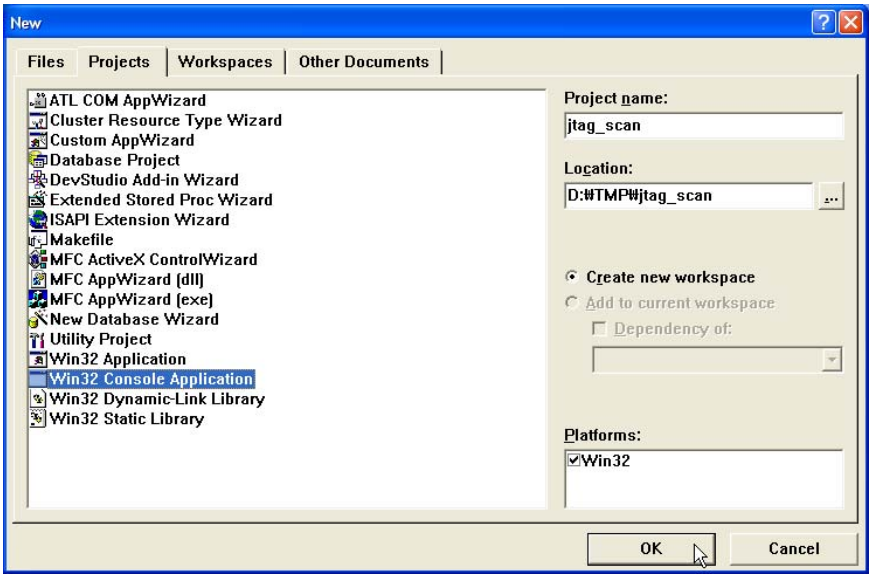
It is assumed you have created the file containing the above main function at ‘D:\tmp\jtag_scan’ directory.

What to do to compile your program for OpenJTAG are two things; one is specifying where OpenJTAG header file locates which will be C:\OpenJTAG\include and the other is specifying OpenJTAG library file which is C:\OpenJTAG\lib\visualc-6.0\open_jtag.lib.

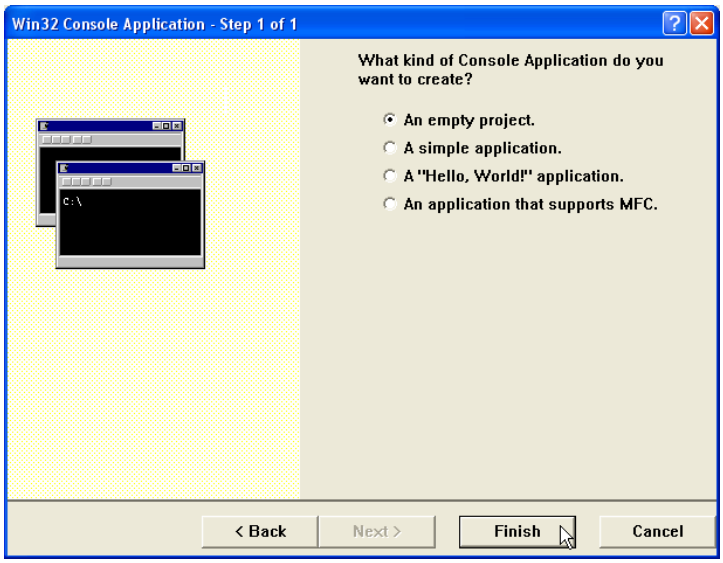
1. Invoke Microsoft Visual C++ studio 6.0
2. Select ‘New..’ from ‘File’ menu.



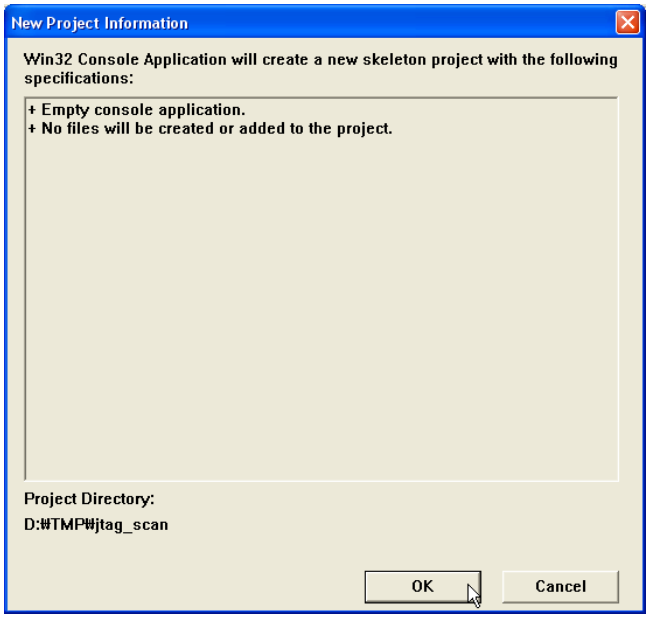
3. Select ‘Win32 Console Application’.
4. Specify ‘Project name:’ as ‘jtag_scan’ or whatever you like.
5. Select ‘Location’ where the main file locates for example C:\OpenJTAG\example\jtag_scan.
6. Then, click ‘OK’ button.



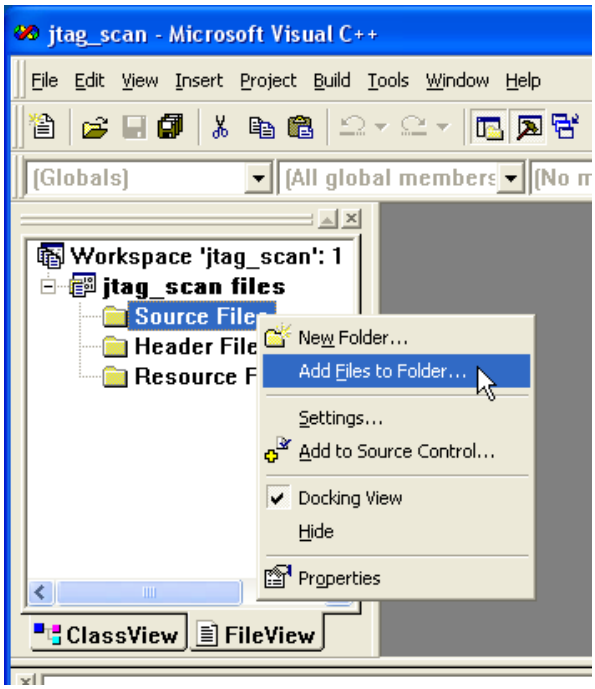
7. Select ‘An empty project’
8. Then, click ‘Finish’ button.


Win32 Console Application - Step 1 of 1
What kind of Console Application do you want to create?
An empty project.
A simple application.
A "Hello, World!" application.
An application that supports MFC.
C:\
< Back
Next >
Finish
Cancel

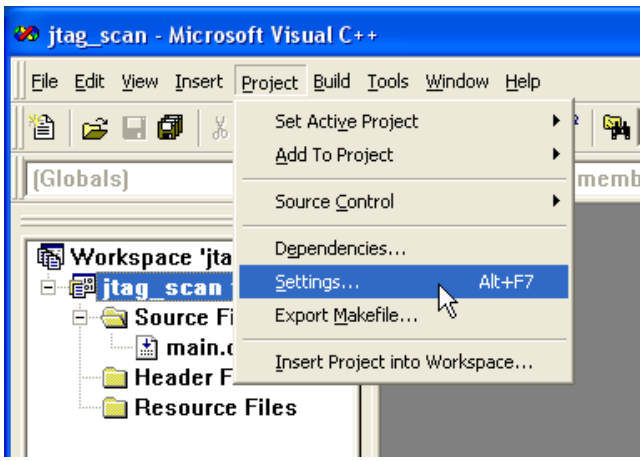
Click ‘OK’ button on the following window.


New Project Information
Win32 Console Application will create a new skeleton project with the following specifications:
+ Empty console application.
+ No files will be created or added to the project.
Project Directory:
D:#TMP#jtag_scan
OK
Cancel

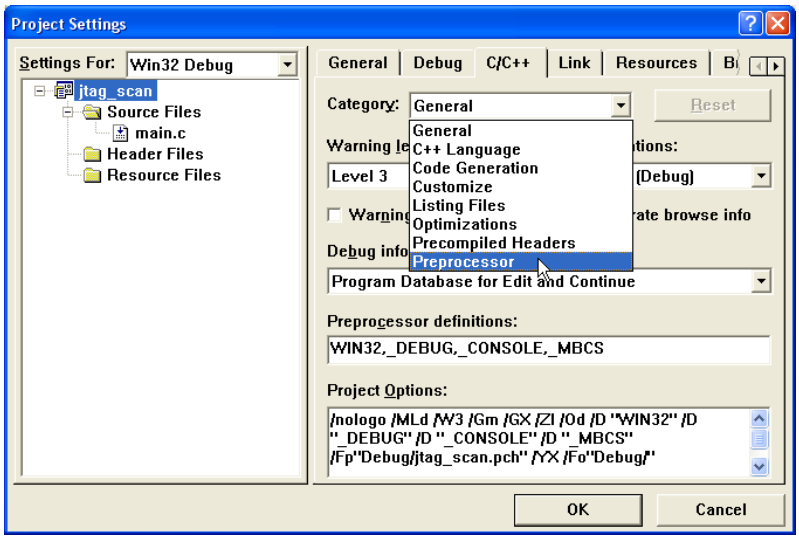
9. Specify the file



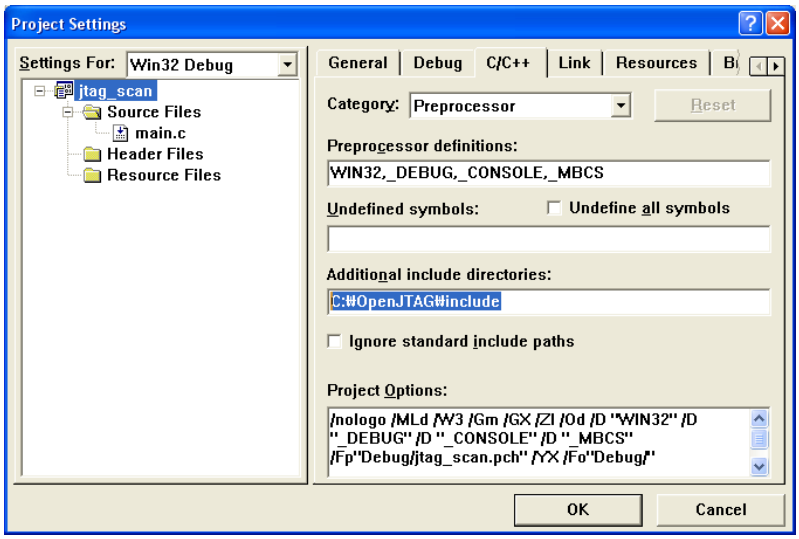
10. Select ‘Settings...’ from ‘Project’ menu.



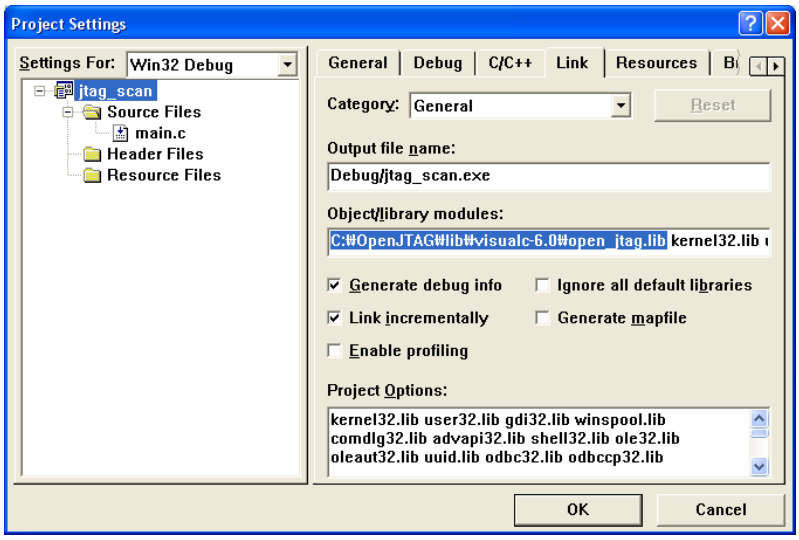
11. Select ‘Preprocessor’ from ‘C/C++’ tab.



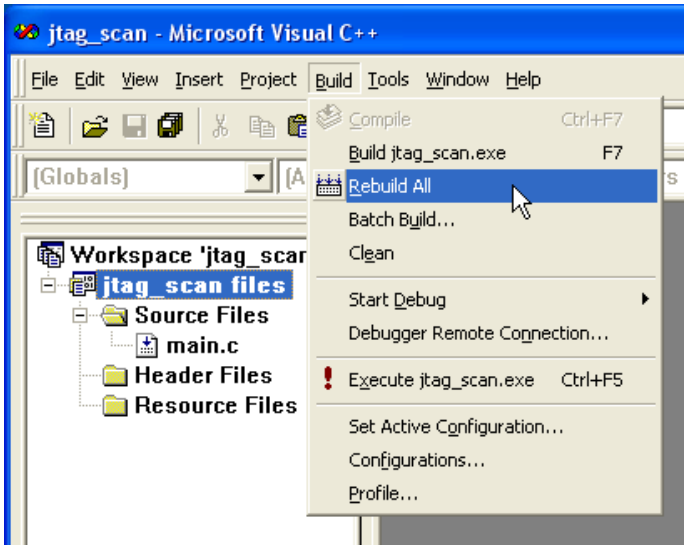
12. Add OpenJTAG include directory at ‘Additional include directories;’, which will be C:\OpenJTAG\include.



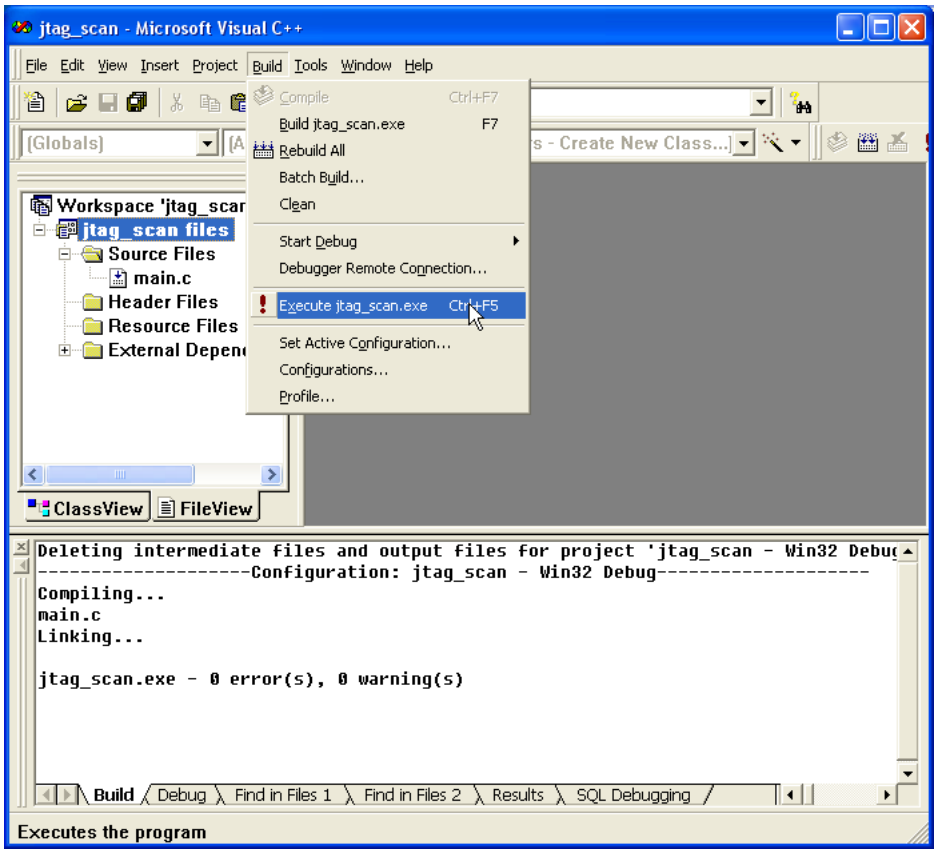
13. Add OpenJTAG library at ‘Object/library modules:’ of ‘Link’ tab, which will be C:\OpenJTAG\lib\visualc-6.0\open_jtag.lib.
14. Then, select ‘OK’ button.



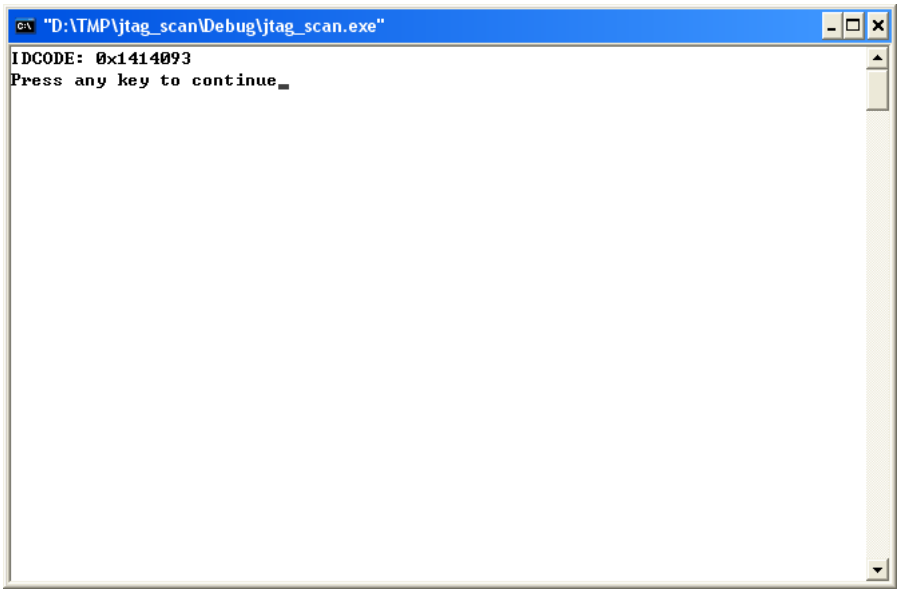
Select ‘Rebuild All’ from ‘Build’ menu to compile the code.



15. Select ‘Execute jtag_scan.exe’ from ‘Build’ menu, but do not forget to connect OpenJTAG to the your computer through USB port and connect a JTAG port to the OpenJTAG.



16. When you see something like the following window, you program runs, but the IDCODE value may differ since it reflect the JTAG device connected to your OpenJTAG.



3.1.3 How to compile with GNU GCC

There is nothing special to compile a program for OpenJTAG, except for specifying where OpenJTAG header file (open_jtag.h) locates and giving OpenJTAG library (libopen_jtag.a) file.

To be added.

dynalith

Chapter 4

OpenJTAG API and Programming

This chapter addresses application programming interface functions for OpenJTAG.

There are three groups of API; bit-wise control group, bit-sequence control group and OpenJTAG control group.

The bit-wise control group API reads and writes JTAG pins, while the bit-sequence control group API generates and receive a number of bit sequence of JTAG pins. The OpenJTAG control group API provides metallization and cleanup process.

4.1 OpenJTAG control API

int oj_init(int config);

'oj_init()' prepares all necessary things to use OpenJTAG so that it should be called before any API calls. When the argument is given as '1', the OpenJTAG hardware is initialized. When the argument is given as '0', the OpenJTAG hardware are used without additional initialization.

int oj_close(void);

'oj_close()' cleans up before terminating the application program so that it is recommended to call this call before end the program.

int oj_set_speed(float clk);

'oj_set_speed()' sets JTAG signal speed in terms of Mhz. For example, when the argument is given as '16.0', OpenJTAG will generates TCK at 16Mhz speed. The maximum speed will be 16Mhz.

4.2 Bit-wise control API

This API enables bit-by-bit control over all JTAG signals including TCK.

int oj_write(unsigned int tall);

'oj_write()' updates all JTAG output pins given by 'tall' argument, which contains {TRSTn, TCK, TMS, TDI, don't care}.

For example 'oj_write(0x12)' updates as follows.

- TRSTn = 1
- TCK = 0
- TMS = 1
- TDI = 0
- TDO = not affected

int oj_read(unsigned int* tall);

'oj_read()' gets value of all JTAG pins.

For example, 'oj_read(&tall)' call after 'oj_write(0x12)' will return 0x12 or 0x13 at 'tall' variable depending on the value of TDO.

4.3 Bit-sequence control API

int oj_cycle_out (unsigned int num, unsigned int trstn, unsigned int tms, unsigned int tdi);

'oj_cycle_out()' generates a number of JTAG cycles specified by 'num', where TRSTn, TMS and TDI pins are driven by the values given at the arguments. It only outputs and does not get value of TDO. The number of cycle can be 32 at most.

Values given at the arguments are driven at JTAG pins lsb (least significant bit) first.

int oj_cycle (unsigned num, unsigned trstn, unsigned tms, unsigned tdi, unsigned *tdo);

'oj_cycle()' generates a number of JTAG cycles specified by 'num', where TRSTn, TMS and TDI pins are driven by the values given at

the arguments. After completion, TDO bit-sequence is stored at the argument ‘tdo’. The number of cycle can be 32 at most.

Values given at the arguments are driven at JTAG pins lsb first and the first value at the TDO pins is stored at the lsb bit of ‘tdo’.

int oj_cycle_wide (unsigned int num, unsigned int *trstn, unsigned int *tms, unsigned int *tdi, unsigned int *tdo);

‘oj_cycle_wide()’ is the same as the ‘oj_cycle()’, but the number of bit-sequence can be up to 128. To support wider than 32-bit, each argument is given by array pointer and the array should be larger than 4.

Values given at the arguments are driven at JTAG pins lsb first and the first value at the TDO pins is stored at the lsb bit of ‘tdo’.

4.3.1 Bit-sequence control example

‘oj_cycle(1, trstn, tms, tdi, &tdo)’ generates JTAG signal as shown in Figure 4-1.

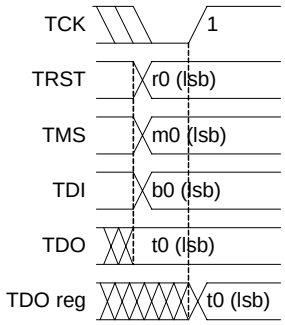


Figure 4-1: **One-cycle operation**

TRSTn, TMS and TDI are driven when TCK goes low. Then, TCK goes high and TDO is read.

‘oj_cycle(n, trstn, tms, tdi, &tdo)’ generates JTAG signal as shown in Figure 4-2.

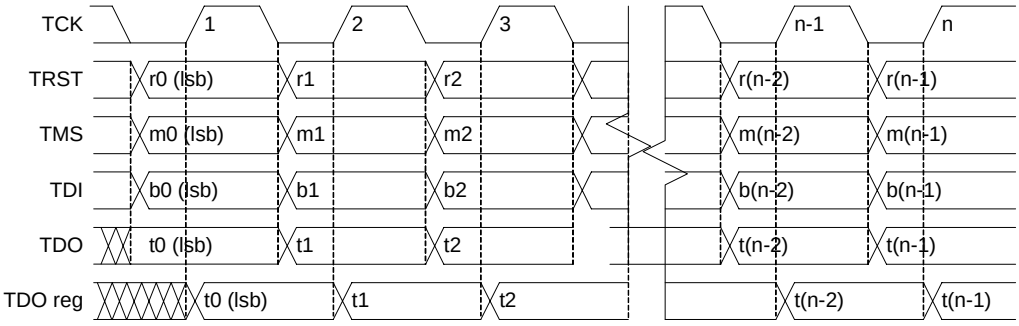


Figure 4-2: **Multi-cycle operation**

4.4 Programming with OpenJTAG API

4.4.1 OpenJTAG header file

OpenJTAG API is defined at ‘open_jtag.h’, which will be found from $(OPENJTAG)/include directory.

4.4.2 OpenJTAG library file

OpenJTAG library is given at the following directories depending on platform and compilers.

Visual Studio 6.0 for Windows XP

$(OPENJTAG)\lib\visualc-6.0\open_jtag.lib

GNU gcc for Linux

$(OPENJTAG)/lib/gcc-xx.yy.zz/libopen_jtag.a

where xx.yy.zz stands for GCC version.

Chapter 5

GDB Server

This chapter describes how to use OpenJTAG for Beyond BA12 processor and OpenRISC processor debugger.

For more information about Beyond BA12 processor, OpenRISC processor and GNU debugger, please refer to the following documents, which are available from http://www.beyondsemi.com, http://www.opencores.org and http://www.gnu.org.

- Beyond Architecture 1 Architecture Manual, Beyond Semiconductor, March 21, 2007.
- BA12 Processor User Manual, Beyond Semiconductor.
- Beyond Semiconductor Debug IF, Beyond Semiconductor, 2007.
- OpenRISC 1000 Architecture Manuel, OPENCORES, November 4, 2005.
- Debugging with GDB, The GNU Source-Level Debugger, 9th Ed., Free Software Foundation, Inc.

5.1 Introduction to overall environment

There are four major components in the context of debugging software along with actual hardware board as shown in Figure 5-1.

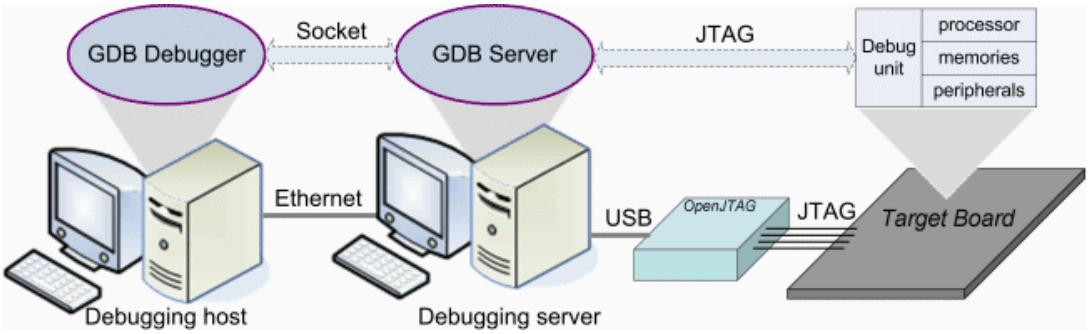


Figure 5-1: **Debugging environment around OpenJTAG**

- Target board contains processor, memory and other peripherals, where the processor should have a proper hardware debugger unit with JTAG port.

- OpenJTAG connects debugger program and the debugger unit in the processor through USB and JTAG.
- GDB server running on the debugging server deals with requests from source-level debugger for the OpenJTAG.
- GDB debugger running on the debugging host debugs the program run by the processor.

GDB server and GDB debugger can be run on different computers over Ethernet, but these two run on the same computer in most cases.

For the case of BA12, the GDB debugger will be 'ba-elf-gdb', which is a GNU GDB ported for BA12 processor and the GDB server will be 'oj-ba12', which is a version for OpenJTAG.

For the case of OpenRISC, the GDB debugger will be 'or32-uclinux-gdb', which is a GNU GDB ported for OpenRISC processor and the GDB server will be 'oj-or12', which is a version for OpenJTAG.

5.2 Prerequisite

It is assumed that the followings are ready to use in your environment.

1. BA12 cross-compilation environment
   - ❑ GNU GCC for BA12
   - ❑ GNU GDB for BA12
1. OpenRISC cross-compilation environment
   - ❑ GNU GCC for OpenRISC
   - ❑ GNU GDB for OpenRISC
2. OpenJTAG software
   - ❑ For installation of OpenJTAG software, please refer to Chapter 2. Installation.

It is assumed that your BA12 Debug IF satisfy the followings. If you are not sure about it, please consult your IP provider or have a look at 'debug_if_defines.v' file.

- ❑ JTAG TAP controller
  - ◆JTAG IDCODE: 0x14951185
- ❑ debug_if
  - ◆HW version: 3
  - ◆Implementation version: 0

Also your OpenRISC Debug IF satisfy the followings. If you are not sure about it, please consult your IP provider or have a look at 'dbg_defines.v' file.

- ❑ JTAG TAP controller
  - ◆JTAG IDCODE: 0xdeadbeef

5.3 Steps to setup

BA12

1. Connect JTAG wires of OpenJTAG to the target board and make sure that JTAG (TCK, TMS, TDI, TDO), VCC and GND are connected correctly.
2. Connect OpenJTAG to the debugging server computer through USB.
3. Run GDB server for BA12 and OpenJTAG, which is 'oj-ba12' in the '$(OPENJTAG)/bin' directory.
4. Run GDB debugger for BA12, which may be 'ba-elf-gdb' in the '$(OPENJTAG)/bin' directory or the one you have in your cross-compilation environment.
5. Then enjoy your BA12 and OpenJTAG.

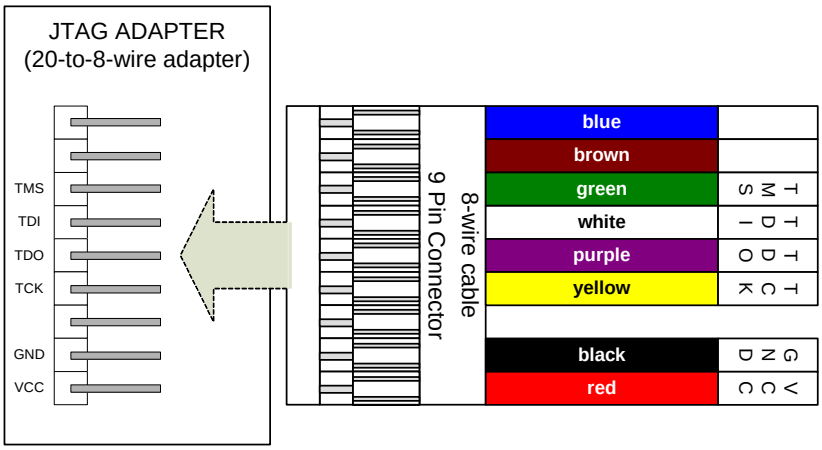
OpenRISC

1. Connect JTAG wires of OpenJTAG to the target board and make sure that JTAG (TCK, TMS, TDI, TDO), VCC and GND are connected correctly.
2. Connect OpenJTAG to the debugging server computer through USB.
3. Run GDB server for OpenRISC and OpenJTAG, which is 'oj-or12' in the '$(OPENJTAG)/bin' directory.
4. Run GDB debugger for OpenRISC, which may be 'or32-uclinux-gdb' in the '$(OPENJTAG)/bin' directory or the one you have in your cross-compilation environment.
5. Then enjoy your OpenRISC and OpenJTAG.

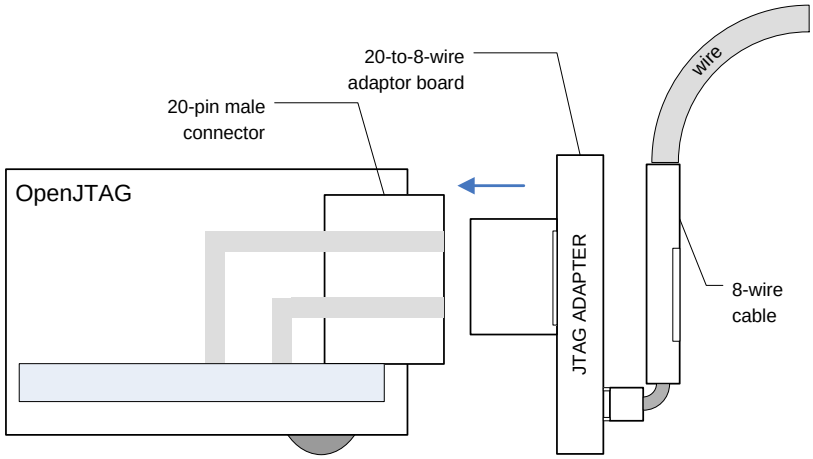
5.3.1 OpenJTAG setup

This step place OpenJTAG between the debugging server computer and the target board.

1. Plug 8-wire cable (9-pin connector) to 20-to-8-wire adapter board.

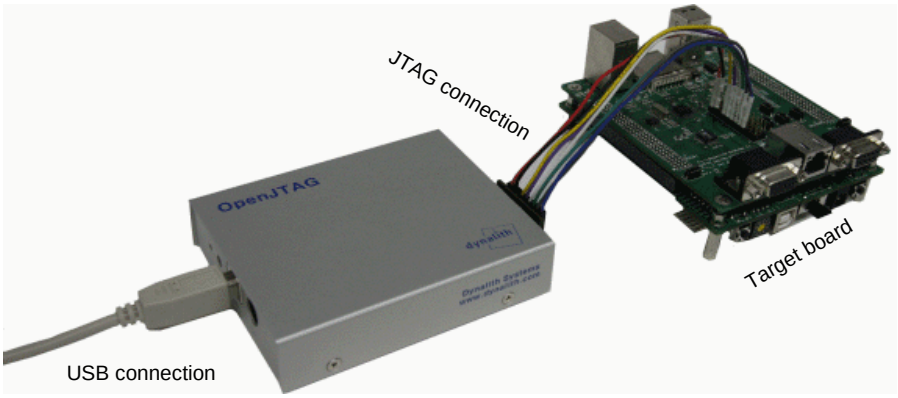


2. Plug the 20-to-8-wire adapter board to 20-pin JTAG connector or OpenJTAG.



3. Plug JTAG wire of 8-wire cable to the JTAG port of the target board.
   - ❑ Red wire goes to VCC
   - ❑ Black wire goes to GND
   - ❑ Yellow wire goes to TCK
   - ❑ Purple wire goes to TDO
   - ❑ White wire goes to TDI
   - ❑ Green wire goes to TMS

4. Connect OpenJTAG to the debugging server computer through USB.



5.3.2 Run GDB server

BA12

GDB server for BA12, oj-ba12 will be found from $(OPENJTAG)/bin directory and it should be run at the computer where OpenJTAG is connected.

The GDB server for BA12, oj-ba12 has various options and these options can be seen by running oj-ba12 with '-h' option.

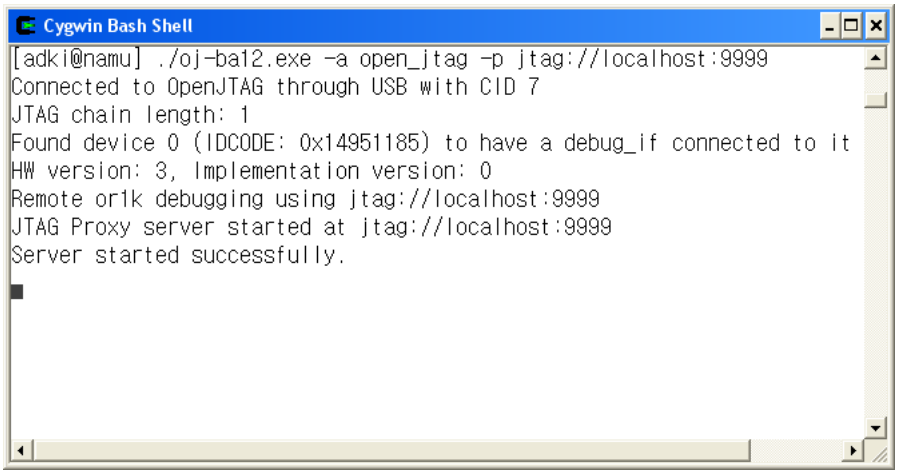
-a open_jtag : specify which cable, ICE is used
-d cid : specify card ID of OpenJTAG
-e eif : specify hardware configuration file of OpenJTAG
-h : print help message
-o : verbose mode on
-p url : specify which computer is used for this server

Most of all, '-a open_jtag' should be used and come first place of options and other options can be committed, since default value will work well.

The GDB server computer is specified by '-p' option with the value of IP address and socket port number. It's default value is 'jtag://localhost:9999', where 'localhost' stands for 'this computer' and '9999' is socket port number.
Thus, typical usage is as follows.

$(OPENJTAG)/bin/oj-ba12 -a open_jtag -p jtag://localhost:9999

The following shows a message after running 'oj-ba12' BA12 GDB server.


Cygwin Bash Shell
[adki@namu] ./oj-ba12.exe -a open_jtag -p jtag://localhost:9999
Connected to OpenJTAG through USB with CID 7
JTAG chain length: 1
Found device 0 (IDCODE: 0x14951185) to have a debug_if connected to it
HW version: 3, Implementation version: 0
Remote or1k debugging using jtag://localhost:9999
JTAG Proxy server started at jtag://localhost:9999
Server started successfully.

OpenRISC

GDB server for OpenRISC, oj-or12 will be found from $(OPEN-JTAG)/bin directory and it should be run at the computer where OpenJTAG is connected.

The GDB server for OpenRISC, oj-or12 has various options and these options can be seen by running oj-or12 with '-h' option.

-a open_jtag : specify which cable, ICE is used
-d cid : specify card ID of OpenJTAG
-e eif : specify hardware configuration file of OpenJTAG
-h : print help message
-o : verbose mode on
-p port : specify which port is used for this server

Most of all, '-a open_jtag' should be used and come first place of options and other options can be committed, since default value will work well.

The GDB server for OpenRISC support localhost connection only. So, '-p port' should be use only socket port number.

Thus, typical usage is as follows.

```
$(OPENJTAG)/bin/oj-or12 -a open_jtag -p 9999
```

CID

The card ID is used to specify the OpenJTAG in the USB bus. It can be 0 ~ 7 and '7' is default value. The card ID is determined by the switch as shown below.

5.3.3 Run GDB debugger

BA12

After the GDB server is running, GDB can be invoked as follows.

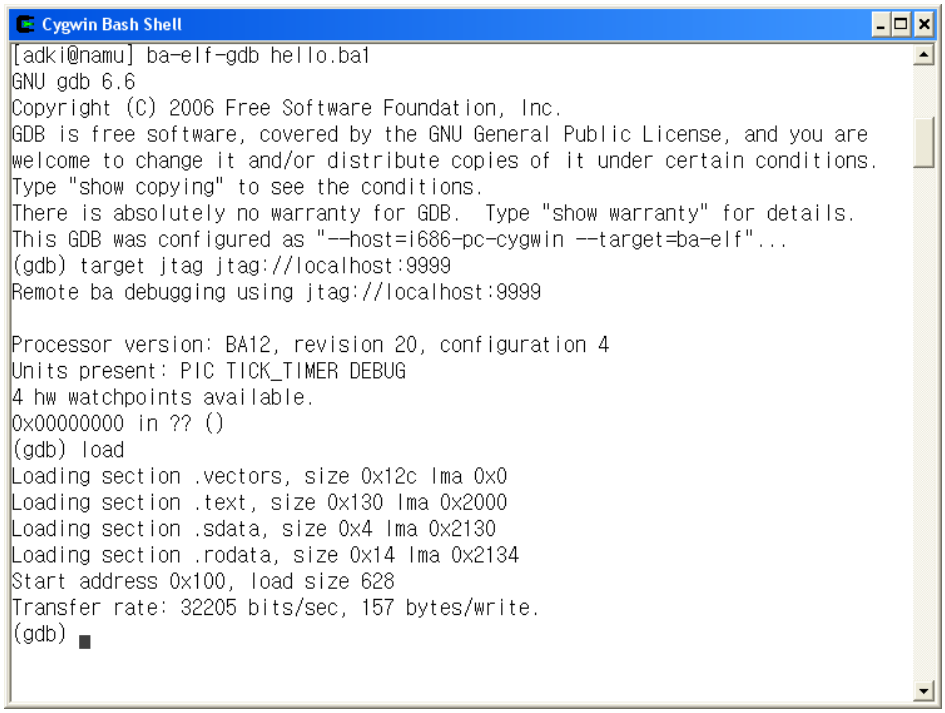
```
ba-elf-gdb  hello.ba1
```

where 'ba-elf-gdb' is GNU GDB for BA12 and 'hello.ba1' is the file name of executable after cross-compiling.

When GDB comes up, issues the following GDB commands.

```
(gdb) target  jtag  jtag://localhost:9999
(gdb) load
(gdb) set $pc=0x100
(gdb) break main
(gdb) continue
```

Where 'target jtag jtag://localhost:9999' command let GDB connect to GDB server at this computer through socket port 9999; 'load' command writes the given program in to the hardware memory; 'set $pc=0x100' command sets PC value to 0x100; 'break main' command sets a break point at the beginning of 'main' function; 'continue' command lets the BA12 processor start at the address specified by the PC.


Cygwin Bash Shell
[adki@namu] ba-elf-gdb hello.ba1
GNU gdb 6.6
Copyright (C) 2006 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty" for details.
This GDB was configured as "--host=i686-pc-cygwin --target=ba-elf"...
(gdb) target jtag jtag://localhost:9999
Remote ba debugging using jtag://localhost:9999
Processor version: BA12, revision 20, configuration 4
Units present: PIC TICK_TIMER DEBUG
4 hw watchpoints available.
0x00000000 in ?? ()
(gdb) load
Loading section .vectors, size 0x12c lma 0x0
Loading section .text, size 0x130 lma 0x2000
Loading section .sdata, size 0x4 lma 0x2130
Loading section .rodata, size 0x14 lma 0x2134
Start address 0x100, load size 628
Transfer rate: 32205 bits/sec, 157 bytes/write.
(gdb)

OpenRISC

After the GDB server is running, GDB can be invoked as follows.

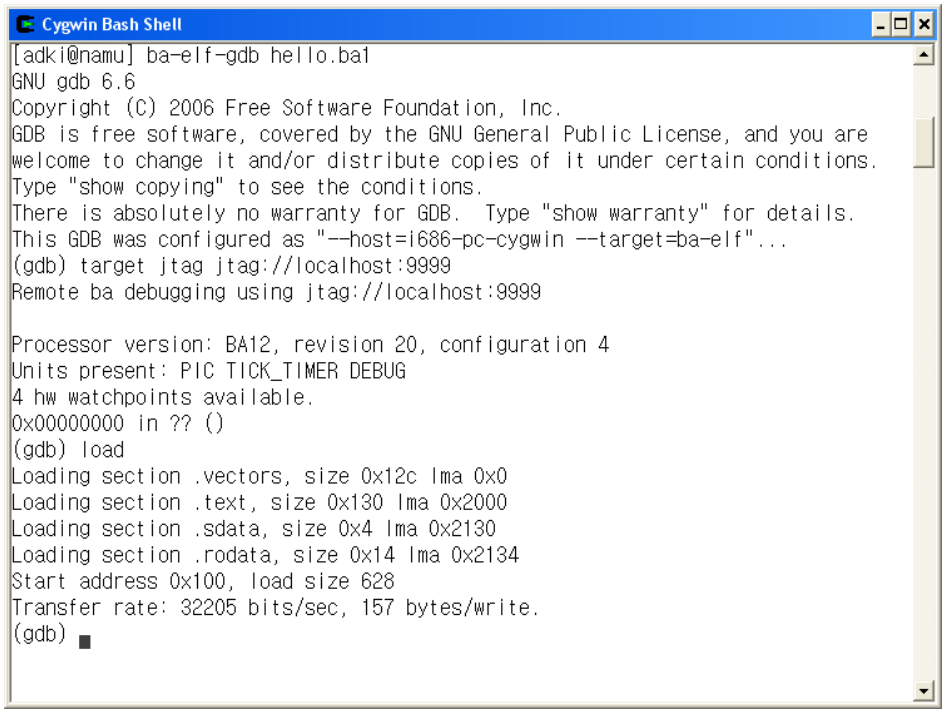
or32-uclinux-gdb hello.or32

where 'or32-uclinux-gdb' is GNU GDB for OpenRISC and 'hello.or32' is the file name of executable after cross-compiling.

When GDB comes up, issues the following GDB commands.

(gdb) target jtag jtag://localhost:9999
(gdb) load
(gdb) set $pc=0x100
(gdb) break main
(gdb) continue

Where 'target jtag jtag://localhost:9999' command let GDB connect to GDB server at this computer through socket port 9999; 'load' command writes the given program in to the hardware memory; 'set $pc=0x100' command sets PC value to 0x100; 'break main' command sets a break point at the beginning of 'main' function; 'continue' command lets the OpenRISC processor start at the address specified by the PC.


Cygwin Bash Shell
[adki@namu] ba-elf-gdb hello.ba1
GNU gdb 6.6
Copyright (C) 2006 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty" for details.
This GDB was configured as "--host=i686-pc-cygwin --target=ba-elf"...
(gdb) target jtag jtag://localhost:9999
Remote ba debugging using jtag://localhost:9999
Processor version: BA12, revision 20, configuration 4
Units present: PIC TICK_TIMER DEBUG
4 hw watchpoints available.
0x00000000 in ?? ()
(gdb) load
Loading section .vectors, size 0x12c lma 0x0
Loading section .text, size 0x130 lma 0x2000
Loading section .sdata, size 0x4 lma 0x2130
Loading section .rodata, size 0x14 lma 0x2134
Start address 0x100, load size 628
Transfer rate: 32205 bits/sec, 157 bytes/write.
(gdb)

Chapter 6

OpenJTAG GUI

This chapter describes how to use OpenJTAG GUI.

6.1 OpenJTAG GUI

OpenJTAG GUI consists of four major parts as shown in Figure 6-1.

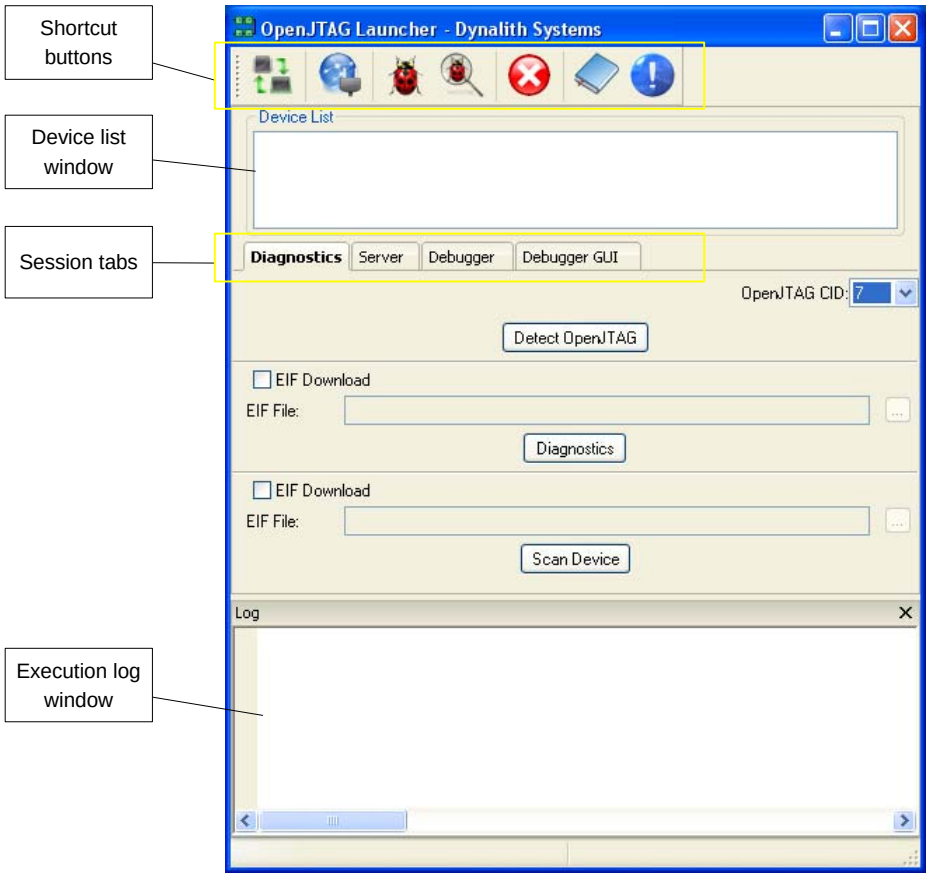


Figure 6-1: **OpenJTAG GUI Layout**

- Shortcut buttons: buttons as shortcuts to access to frequently used functions
- Device list window: display JTAG device list after 'Scan Device'
- Session tabs:

- Execution log window: display execution results

6.2 Diagnostic tab

Through this tab, various functions can be done including OpenJTAG hardware detection, JTAG device detection and so on.

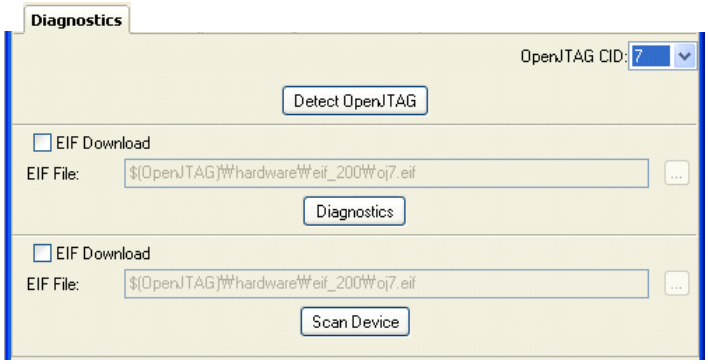


Figure 6-2: **OpenJTAG Diagnostics tab**

6.2.1 Detect OpenJTAG

When 'Detect OpenJTAG' button is selected, USB is scanned and displays OpenJTAG hardware if any.

6.2.2 Diagnostics

When 'Diagnostics' button is selected, the detected OpenJTAG hardware is tested.

When 'EIF Download' box is checked, the specified EIF[1] is downloaded before diagnostics is carried out. For normal case, it is not necessary.

6.2.3 Scan Device

When 'Scan Device' button is selected, JTAG device is scanned and the results is displayed at 'Device List' window.

1. EIF stands for Emulation Information File and is a sort of hardware configuration data.

When ‘EIF Download’ box is checked, the specified EIF[1] is downloaded before diagnostics is carried out. For normal case, it is not necessary.

6.3 Server tab

Through this tab, debugger server, i.e., GDB Server, can be invoked.

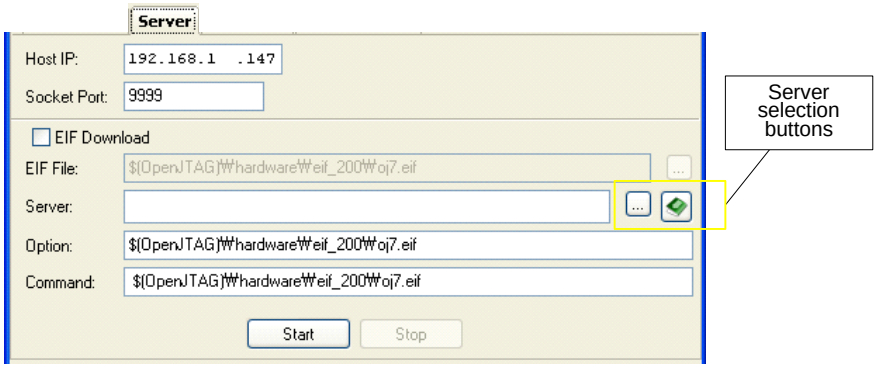


Figure 6-3: **OpenJTAG Server tab**

6.3.1 Host IP field

This field specify which computer is used to run debugger server. Current local host is selected automatically.

Note that OpenJTAG GUI should be run on the computer where debugger server is run.

6.3.2 Socket Port field

This field specify which socket port is used to debugger server. ‘9999’ is used by default.

Note that care should be taken when you change socket port number since some ports are already used for other services.

1. EIF stands for Emulation Information File and is a sort of hardware configuration data.

6.3.3 Server field

This field specify which debugger server program is used.

By selecting sever selection button, ready-to-run debugger server can be selected or user's own debugger server can be also selected.

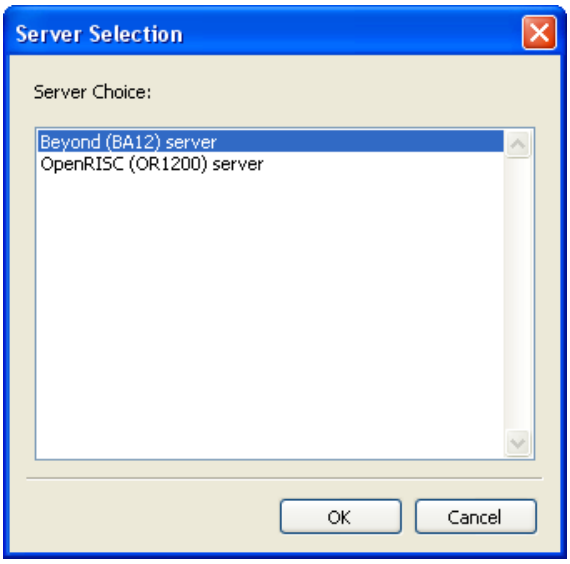


Figure 6-4: **Ready-to-run debugger server list**

- Beyond (BA12) server
  - Note that the current version of Beyond (BA12) server is optimized for a single JTAG tab.
- OpenRISC (OR1200) server

6.3.4 Start and stop button

When 'Start' button is selected, the specified server runs.

When 'Stop' button is selected, the specified server and its related processes are killed.

6.4 Debugger tab

Through this tab, debugger, i.e., GDB, can be invoked.

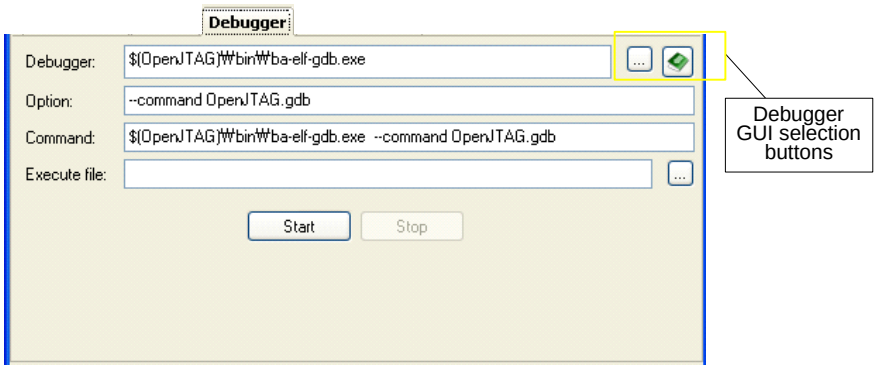


Figure 6-5: **OpenJTAG Debugger tab**

6.4.1 Debugger field

This field specify which debugger program is used.

By selecting debugger selection button, ready-to-run debugger can be selected or user's own debugger can be also selected.

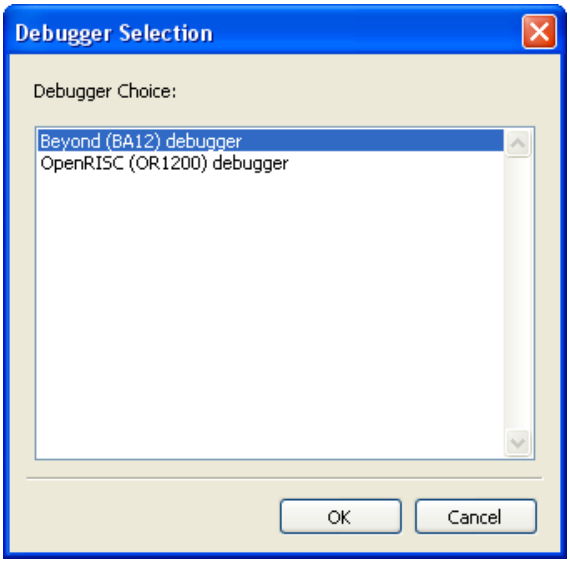


Figure 6-6: **Ready-to-run debugger**

6.4.2 Option field

Debugger specific options can be provided through this field.

6.4.3 Command field

This field is filled automatically by the contents of debugger and option field.

6.4.4 Execute file field

This filed specify which program, i.e., debuggee program, to be debugged.

6.4.5 Start and stop button

When ‘Start’ button is selected, the specified debugger runs.

When ‘Stop’ button is selected, the specified debugger and its related processes are killed.

6.5 Debugger GUI tab

Through this tab, debugger GUI, i.e., DDD, can be invoked.

Note that this GUI uses the debugger specified at ‘Debugger tab’ and debuggee program specified at ‘Debugger tab’.

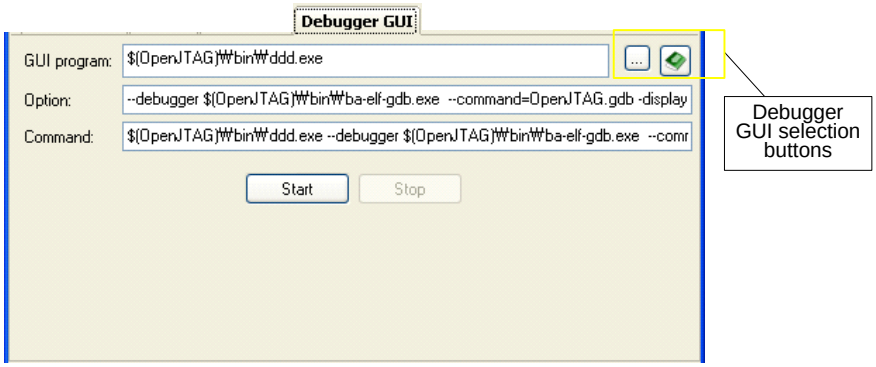


Figure 6-7: **OpenJTAG Debugger GUI tab**

6.5.1 GUI program field

This field specify which debugger GUI program is used.

By selection debugger GUI selection button, ready-to-run debugger GUI can be selected or user's own debugger GUI can be also selected.

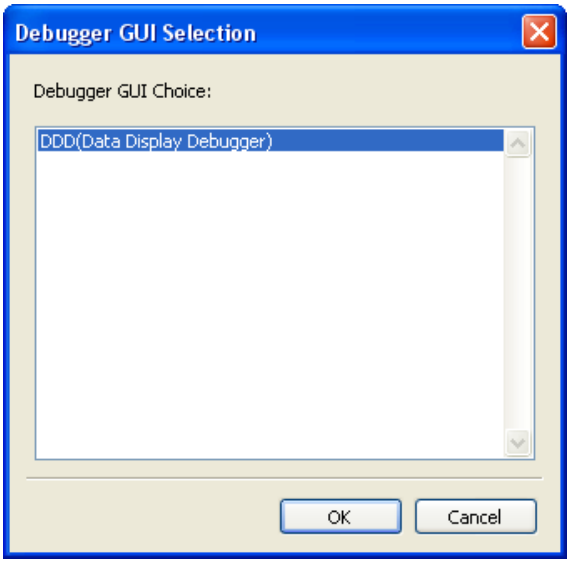


Figure 6-8: **Ready-to-run debugger GUI**

6.5.2 Option field

Debugger GUI specific options can be provided through this field.

6.5.3 Command field

This field is filled automatically by the contents of debugger and option field.

6.5.4 Start and stop button

When 'Start' button is selected, the specified GUI runs.

When 'Stop' button is selected, the specified GUI and its related processes are killed.

6.6 How to use OpenJTAG for BA12

It is quite easy to use OpenJTAG GUI for BA12. As shown in Figure 6-9, GDB server (oj-ba12) and GDB debugger (ba-elf-gdb) should be run along with OpenJTAG.

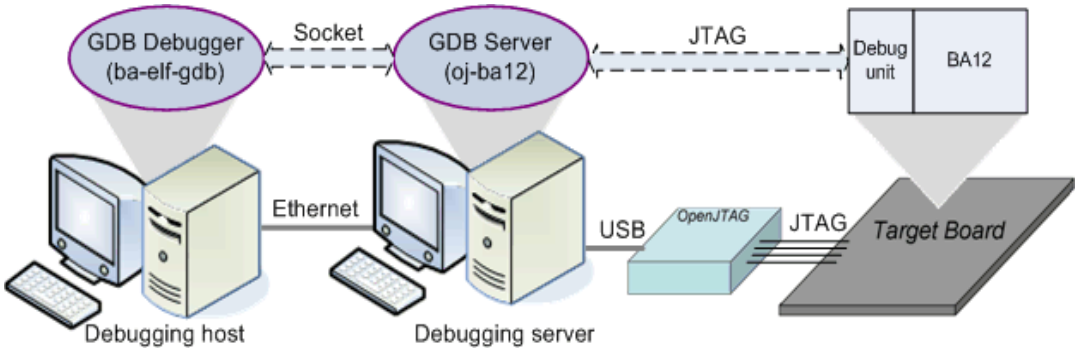


Figure 6-9: **Debugging environment with BA12**

1. Prepare debuggee program, i.e., executable for BA12 using BA12 cross-compiler.
2. Connect OpenJTAG to the target board through JTAG
3. Connect OpenJTAG to the debugger server, which is where OpenJTAG runs.
4. Invoke OpenJTAG on the debugger server.
5. 'Detect OpenJTAG' on 'Diagnostic tab'.
6. 'Scan Device' on 'Diagnostic tab'.
7. Select 'Beyond (BA12) server' from 'ready-to-run debugger server list' on 'Server tab'.
8. 'Start' on 'Server tab'.
9. Select 'Beyond (BA12) debugger' from 'ready-to-run debugger list' on 'Debugger tab'.
10. 'Start' on 'Debugger tab'.

Now enjoy your debugging.

6.7 How to use OpenJTAG for OpenRISC

It is quite easy to use OpenJTAG GUI for OpenRISC. As shown in Figure 6-10, GDB server (oj-or12) and GDB debugger (or32-uclinux-gdb) should be run along with OpenJTAG.

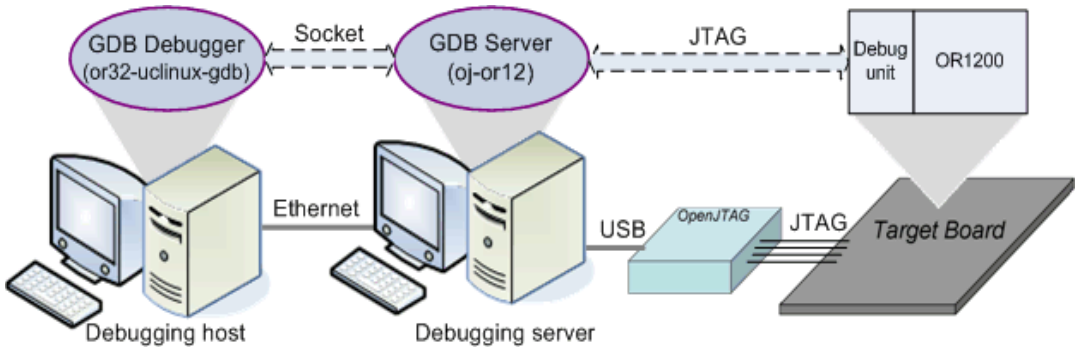


Figure 6-10: **Debugging environment with OpenRISC**

1. Prepare debuggee program, i.e., executable for OpenRISC using OpenRISC cross-compiler.
2. Connect OpenJTAG to the target board through JTAG
3. Connect OpenJTAG to the debugger server, which is where OpenJTAG runs.
4. Invoke OpenJTAG on the debugger server.
5. 'Detect OpenJTAG' on 'Diagnostic tab'.
6. 'Scan Device' on 'Diagnostic tab'.
7. Select 'OpenRISC (OR1200) server' from 'ready-to-run debugger server list' on 'Server tab'.
8. 'Start' on 'Server tab'.
9. Select 'OpenRISC (OR1200) debugger' from 'ready-to-run debugger list' on 'Debugger tab'.
10. 'Start' on 'Debugger tab'.

Now enjoy your debugging.

Index

Numerics

20-to-8-wire adapter board 5-4
8-wire cable 5-4
9-pin connector 5-4

B

Boundary Scan 1-1
BYPASS 3-1

C

card ID 5-7

D

debug_if 5-2
debug_if_defines.v 5-2, 5-3
debugging host 5-2
debugging server 5-2

E

Environment variable 2-3

G

GDB debugger 5-2
GDB server 5-2

I

icon 2-3
IDCODE 3-1, 5-2, 5-3
include 3-3, 4-4

J

Joint Test Action Group. See JTAG
JTAG 1-1
jtag://localhost:9999 5-5

L

libopen_jtag.a 3-10, 4-4
localhost 5-5

O

oj_close() 3-3, 4-1
oj_cycle() 3-2, 4-2
oj_cycle_out() 4-2
oj_cycle_wide() 4-3
oj_init() 3-3, 4-1
oj_read() 4-2
oj_set_speed() 4-1
oj_write() 4-2
open_jtag.h 3-3, 3-10, 4-4
open_jtag.lib 3-3, 3-7, 4-4
OPENJTAG 2-3
OpenJTAG 1-1

S

socket port 5-5, 5-6

T

TAP 1-1, 3-1
TAP controller 3-1
target board 5-1
Test Access Port. See TAP